下载APP

# 13 | GEO是什么？还可以定义新的数据类型吗？

2020-09-04 蒋德钧

Redis核心技术与实战

进入课程 >

**讲述：蒋德钧**

时长 20:14 大小 18.54M

你好，我是蒋德钧。

在第 2 讲中，我们学习了 Redis 的 5 大基本数据类型：String、List、Hash、Set 和 Sorted Set，它们可以满足大多数的数据存储需求，但是在面对海量数据统计时，它们的内存开销很大，而且对于一些特殊的场景，它们是无法支持的。所以，Redis 还提供了 3 种扩展数据类型，分别是 Bitmap、HyperLogLog 和 GEO。前两种我在上节课已经重点介绍过了，今天，我再具体讲一讲 GEO。

另外，我还会给你介绍开发自定义的新数据类型的基本步骤。掌握了自定义数据类型的开发方法，当你面临一些复杂的场景时，就不用受基本数据类型的限制，可以直接在 Redis 中增加定制化的数据类型，来满足你的特殊需求。

接下来，我们就先来了解下扩展数据类型 GEO 的实现原理和使用方法。

## 面向 LBS 应用的 GEO 数据类型

在日常生活中，我们越来越依赖搜索“附近的餐馆”、在打车软件上叫车，这些都离不开基于位置信息服务（Location-Based Service，LBS）的应用。LBS 应用访问的数据是和人或物关联的一组经纬度信息，而且要能查询相邻的经纬度范围，GEO 就非常适合应用在 LBS 服务的场景中，我们来看一下它的底层结构。

## GEO 的底层结构

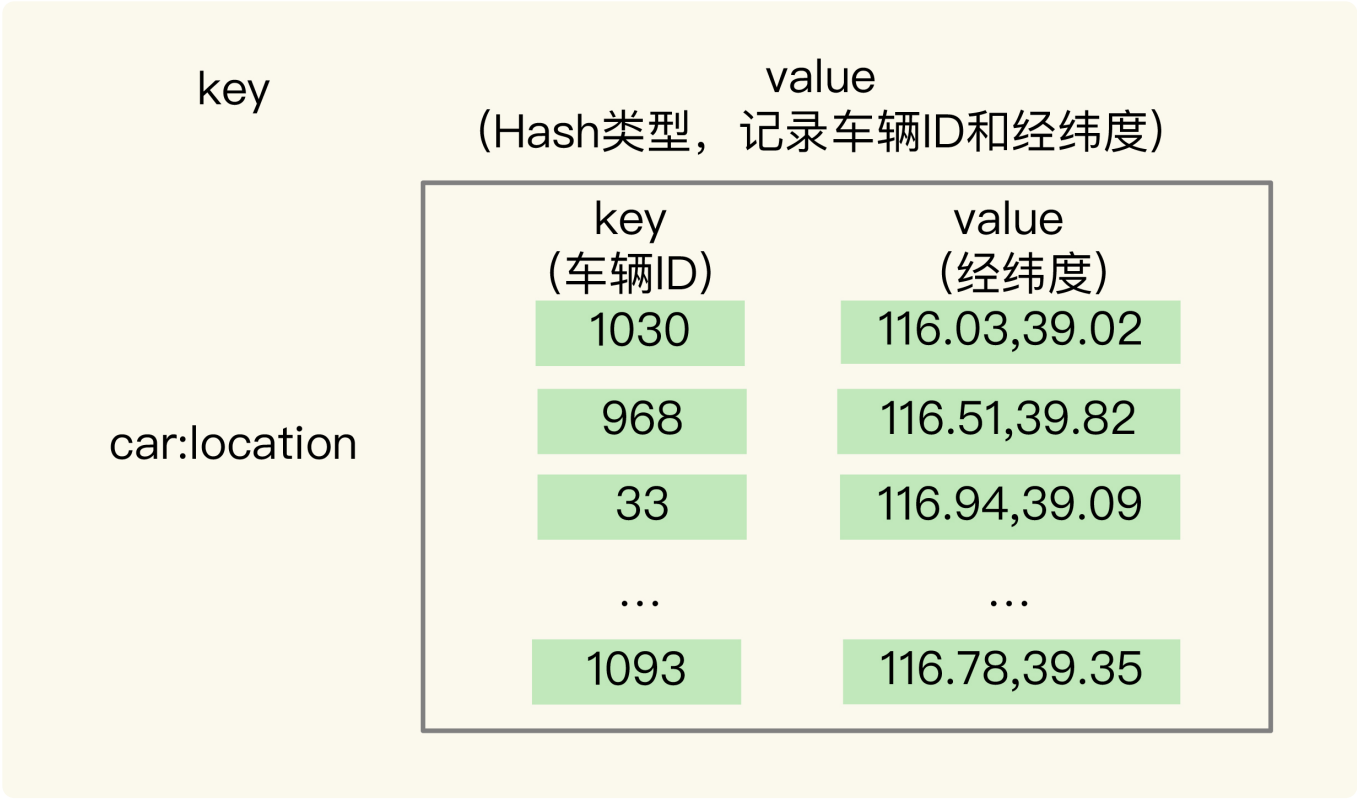
**一般来说，在设计一个数据类型的底层结构时，我们首先需要知道，要处理的数据有什么访问特点。**所以，我们需要先搞清楚位置信息到底是怎么被存取的。

我以叫车服务为例，来分析下 LBS 应用中经纬度的存取特点。

1. 每一辆网约车都有一个编号（例如 33），网约车需要将自己的经度信息（例如 116.034579）和纬度信息（例如 39.000452 ）发给叫车应用。
2. 用户在叫车的时候，叫车应用会根据用户的经纬度位置（例如经度 116.054579，纬度 39.030452），查找用户的附近车辆，并进行匹配。
3. 等把位置相近的用户和车辆匹配上以后，叫车应用就会根据车辆的编号，获取车辆的信息，并返回给用户。

可以看到，一辆车（或一个用户）对应一组经纬度，并且随着车（或用户）的位置移动，相应的经纬度也会变化。

这种数据记录模式属于一个 key（例如车 ID）对应一个 value（一组经纬度）。当有很多车辆信息要保存时，就需要有一个集合来保存一系列的 key 和 value。Hash 集合类型可以快速存取一系列的 key 和 value，正好可以用来记录一系列车辆 ID 和经纬度的对应关系，所以，我们可以把不同车辆的 ID 和它们对应的经纬度信息存在 Hash 集合中，如下图所示：




同时，Hash 类型的 HSET 操作命令，会根据 key 来设置相应的 value 值，所以，我们可以用它来快速地更新车辆变化的经纬度信息。

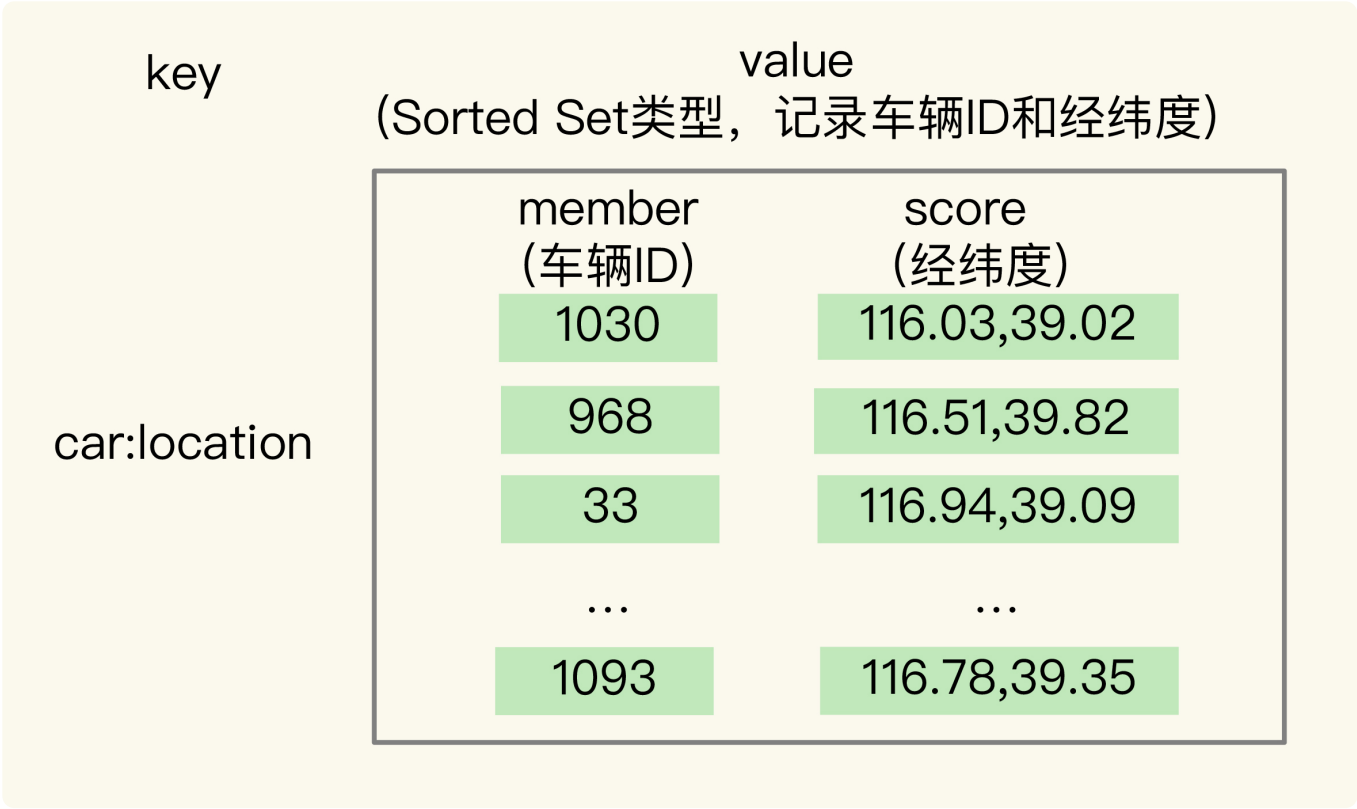
到这里，Hash 类型看起来是一个不错的选择。但问题是，对于一个 LBS 应用来说，除了记录经纬度信息，还需要根据用户的经纬度信息在车辆的 Hash 集合中进行范围查询。一旦涉及到范围查询，就意味着集合中的元素需要有序，但 Hash 类型的元素是无序的，显然不能满足我们的要求。

我们再来看看使用 **Sorted Set 类型**是不是合适。

Sorted Set 类型也支持一个 key 对应一个 value 的记录模式，其中，key 就是 Sorted Set 中的元素，而 value 则是元素的权重分数。更重要的是，Sorted Set 可以根据元素的权重分数排序，支持范围查询。这就能满足 LBS 服务中查找相邻位置的需求了。

实际上，GEO 类型的底层数据结构就是用 Sorted Set 来实现的。咱们还是借着叫车应用的例子来加深下理解。

用 Sorted Set 来保存车辆的经纬度信息时，Sorted Set 的元素是车辆 ID，元素的权重分数是经纬度信息，如下图所示：




这时问题来了，Sorted Set 元素的权重分数是一个浮点数（float 类型），而一组经纬度包含的是经度和纬度两个值，是没法直接保存为一个浮点数的，那具体该怎么进行保存呢？

这就要用到 GEO 类型中的 GeoHash 编码了。

## GeoHash 的编码方法

为了能高效地对经纬度进行比较，Redis 采用了业界广泛使用的 GeoHash 编码方法，这个方法的基本原理就是“二分区间，区间编码”。

当我们要对一组经纬度进行 GeoHash 编码时，我们要先对经度和纬度分别编码，然后再把经纬度各自的编码组合成一个最终编码。

首先，我们来看下经度和纬度的单独编码过程。

对于一个地理位置信息来说，它的经度范围是[-180,180]。GeoHash 编码会把一个经度值编码成一个 N 位的二进制值，我们来对经度范围[-180,180]做 N 次的二分区操作，其中 N 可以自定义。

在进行第一次二分区时，经度范围[-180,180]会被分成两个子区间：[-180,0) 和[0,180]（我称之为左、右分区）。此时，我们可以查看一下要编码的经度值落在了左分区还是右分区。如果是落在左分区，我们就用 0 表示；如果落在右分区，就用 1 表示。这样一来，每做完一次二分区，我们就可以得到 1 位编码值。

然后，我们再对经度值所属的分区再做一次二分区，同时再次查看经度值落在了二分区后的左分区还是右分区，按照刚才的规则再做 1 位编码。当做完 N 次的二分区后，经度值就可以用一个 N bit 的数来表示了。

举个例子，假设我们要编码的经度值是 116.37，我们用 5 位编码值（也就是 N=5，做 5 次分区）。

我们先做第一次二分区操作，把经度区间[-180,180]分成了左分区[-180,0) 和右分区[0,180]，此时，经度值 116.37 是属于右分区[0,180]，所以，我们用 1 表示第一次二分区后的编码值。

接下来，我们做第二次二分区：把经度值 116.37 所属的[0,180]区间，分成[0,90) 和[90, 180]。此时，经度值 116.37 还是属于右分区[90,180]，所以，第二次分区后的编码值仍然为 1。等到第三次对[90,180]进行二分区，经度值 116.37 落在了分区后的左分区[90, 135)中，所以，第三次分区后的编码值就是 0。

按照这种方法，做完 5 次分区后，我们把经度值 116.37 定位在[112.5, 123.75]这个区间，并且得到了经度值的 5 位编码值，即 11010。这个编码过程如下表所示：

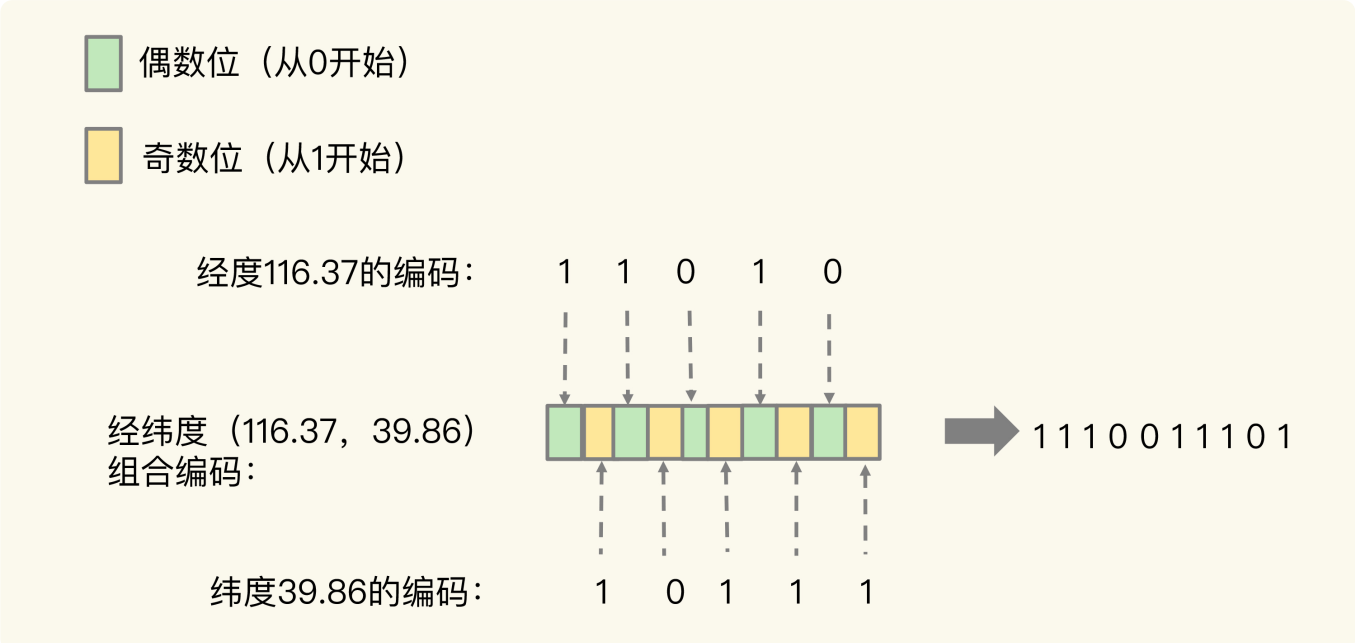
| 分区次数 | 最小经度值 | 二分后的中间值 | 最大经度值 | 经度116.37所在区间 | 经度的GeoHash编码 |
|---|---|---|---|---|---|
| 第一次 | -180 | 0 | 180 | [0, 180] | 1 |
| 第二次 | 0 | 90 | 180 | [90, 180] | 1 |
| 第三次 | 90 | 135 | 180 | [90, 135) | 0 |
| 第四次 | 90 | 112.5 | 135 | [112.5, 135] | 1 |
| 第五次 | 112.5 | 123.75 | 135 | [112.5, 123.75] | 0 |

对纬度的编码方式，和对经度的一样，只是纬度的范围是[-90，90]，下面这张表显示了对纬度值 39.86 的编码过程。

| 分区次数 | 最小纬度值 | 二分后的中间值 | 最大纬度值 | 纬度39.86所在区间 | 纬度的GeoHash编码 |
|---|---|---|---|---|---|
| 第一次 | -90 | 0 | 90 | [0, 90] | 1 |
| 第二次 | 0 | 45 | 90 | [0, 45) | 0 |
| 第三次 | 0 | 22.5 | 45 | [22.5, 45] | 1 |
| 第四次 | 22.5 | 33.75 | 45 | [33.75, 45] | 1 |
| 第五次 | 33.75 | 39.375 | 45 | [39.375, 45] | 1 |

当一组经纬度值都编完码后，我们再把它们的各自编码值组合在一起，组合的规则是：最终编码值的偶数位上依次是经度的编码值，奇数位上依次是纬度的编码值，其中，偶数位从 0 开始，奇数位从 1 开始。

我们刚刚计算的经纬度（116.37，39.86）的各自编码值是 11010 和 10111，组合之后，第 0 位是经度的第 0 位 1，第 1 位是纬度的第 0 位 1，第 2 位是经度的第 1 位 1，第 3 位是纬度的第 1 位 0，以此类推，就能得到最终编码值 1110011101，如下图所示：




用了 GeoHash 编码后，原来无法用一个权重分数表示的一组经纬度（116.37，39.86）就可以用 1110011101 这一个值来表示，就可以保存为 Sorted Set 的权重分数了。

当然，使用 GeoHash 编码后，我们相当于把整个地理空间划分成了一个个方格，每个方格对应了 GeoHash 中的一个分区。

举个例子。我们把经度区间[-180,180]做一次二分区，把纬度区间[-90,90]做一次二分区，就会得到 4 个分区。我们来看下它们的经度和纬度范围以及对应的 GeoHash 组合编码。

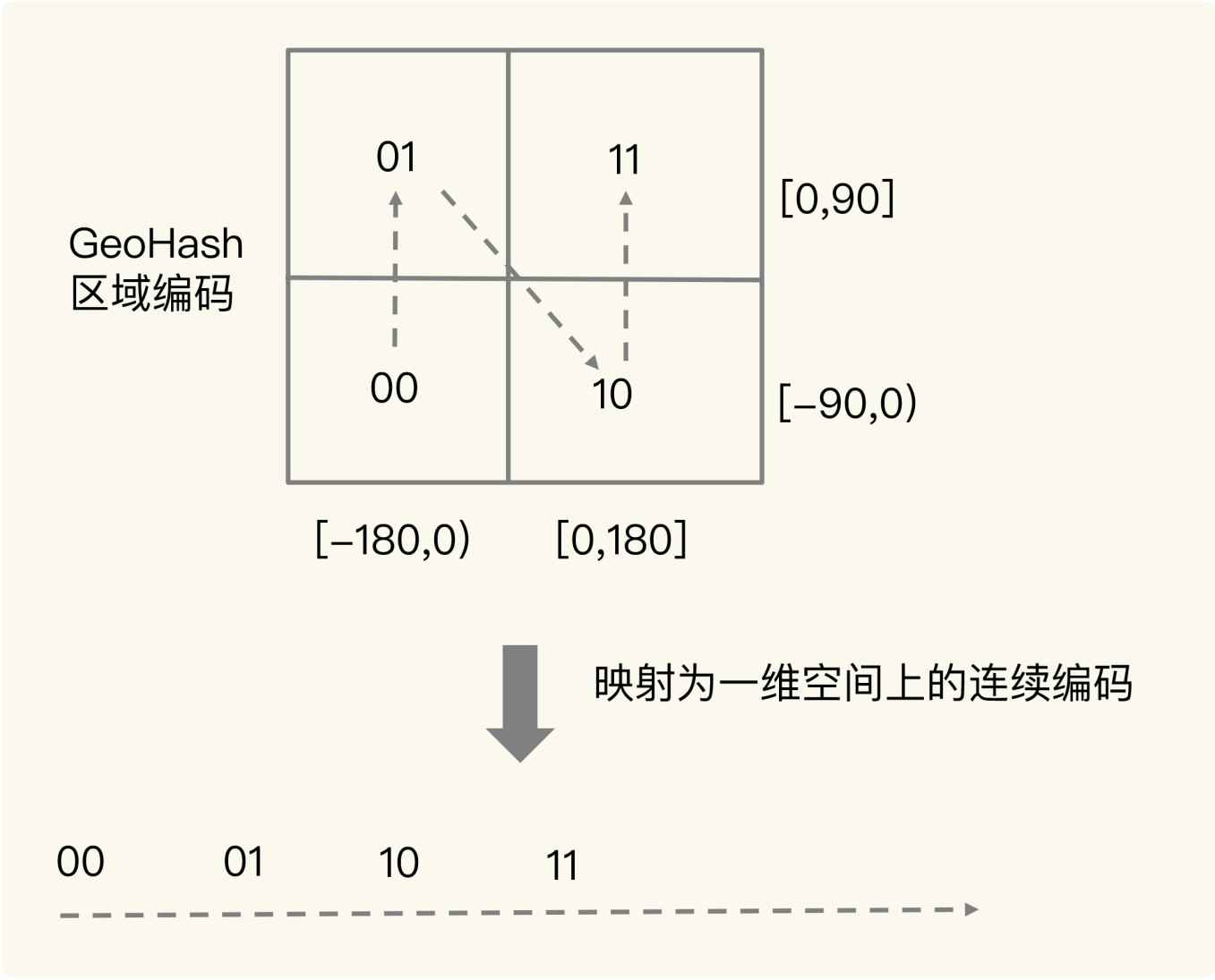
分区一：[-180,0) 和[-90,0)，编码 00；

分区二：[-180,0) 和[0,90]，编码 01；

分区三：[0,180]和[-90,0)，编码 10；

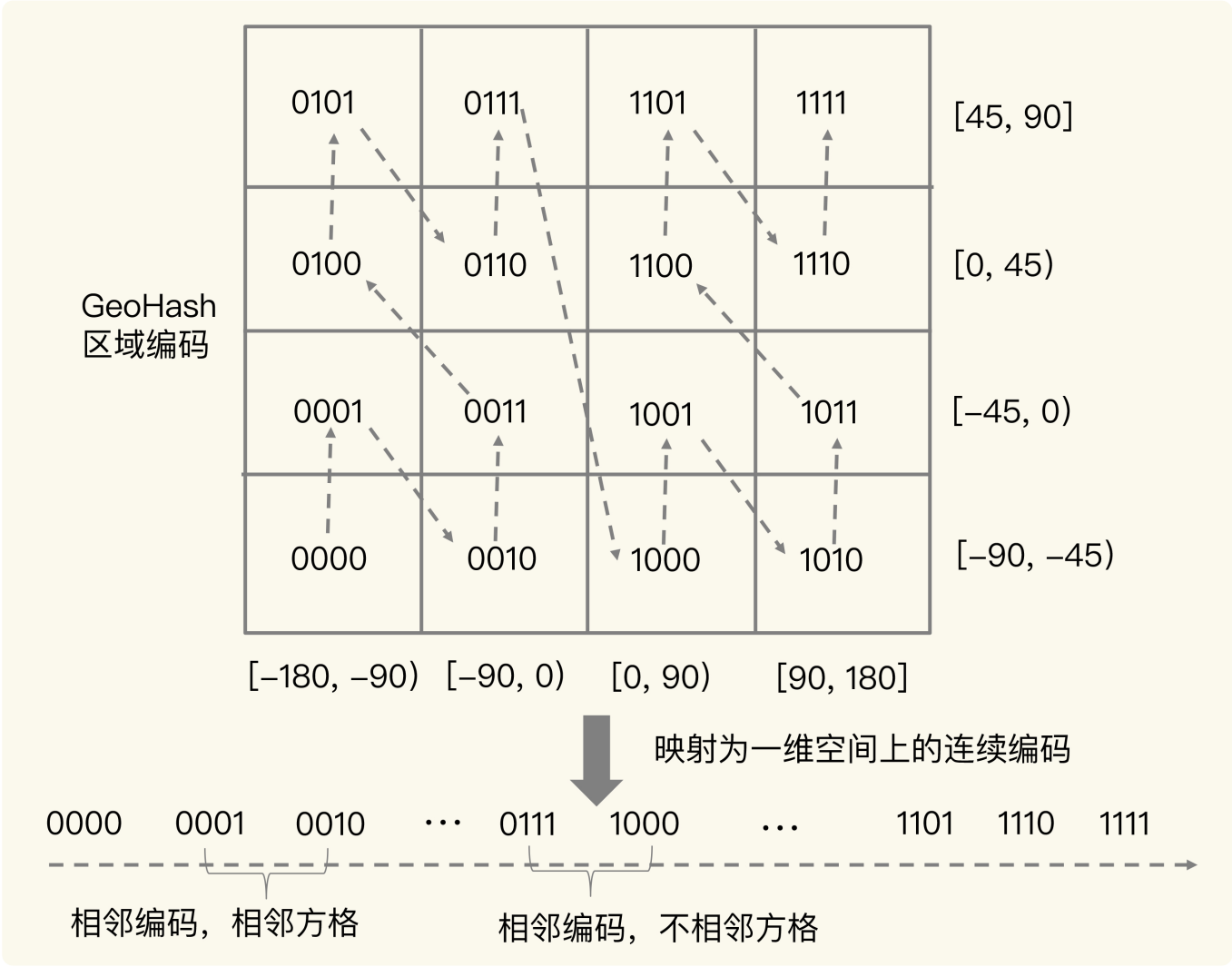
分区四：[0,180]和[0,90]，编码 11。

这 4 个分区对应了 4 个方格，每个方格覆盖了一定范围内的经纬度值，分区越多，每个方格能覆盖到的地理空间就越小，也就越精准。我们把所有方格的编码值映射到一维空间时，相邻方格的 GeoHash 编码值基本也是接近的，如下图所示：




所以，我们使用 Sorted Set 范围查询得到的相近编码值，在实际的地理空间上，也是相邻的方格，这就可以实现 LBS 应用“搜索附近的人或物”的功能了。

不过，我要提醒你一句，有的编码值虽然在大小上接近，但实际对应的方格却距离比较远。例如，我们用 4 位来做 GeoHash 编码，把经度区间[-180,180]和纬度区间[-90,90]各分成了 4 个分区，一共 16 个分区，对应了 16 个方格。编码值为 0111 和 1000 的两个方格就离得比较远，如下图所示：




所以，为了避免查询不准确问题，我们可以同时查询给定经纬度所在的方格周围的 4 个或 8 个方格。

好了，到这里，我们就知道了，GEO 类型是把经纬度所在的区间编码作为 Sorted Set 中元素的权重分数，把和经纬度相关的车辆 ID 作为 Sorted Set 中元素本身的值保存下来，这样相邻经纬度的查询就可以通过编码值的大小范围查询来实现了。接下来，我们再来聊聊具体如何操作 GEO 类型。

## 如何操作 GEO 类型?

在使用 GEO 类型时，我们经常会用到两个命令，分别是 GEOADD 和 GEORADIUS。

GEOADD 命令：用于把一组经纬度信息和相对应的一个 ID 记录到 GEO 类型集合中；

GEORADIUS 命令：会根据输入的经纬度位置，查找以这个经纬度为中心的一定范围内的其他元素。当然，我们可以自己定义这个范围。

我还是以叫车应用的车辆匹配场景为例，介绍下具体如何使用这两个命令。

假设车辆 ID 是 33，经纬度位置是（116.034579，39.030452），我们可以用一个 GEO 集合保存所有车辆的经纬度，集合 key 是 cars:locations。执行下面的这个命令，就可以把 ID 号为 33 的车辆的当前经纬度位置存入 GEO 集合中：

复制代码

```
GEOADD cars:locations 116.034579 39.030452 33
```

当用户想要寻找自己附近的网约车时，LBS 应用就可以使用 GEORADIUS 命令。

例如，LBS 应用执行下面的命令时，Redis 会根据输入的用户的经纬度信息（116.054579，39.030452 ），查找以这个经纬度为中心的 5 公里内的车辆信息，并返回给 LBS 应用。当然， 你可以修改“5”这个参数，来返回更大或更小范围内的车辆信息。

复制代码

```
GEORADIUS cars:locations 116.054579 39.030452 5 km ASC COUNT 10
```

另外，我们还可以进一步限定返回的车辆信息。

比如，我们可以使用 ASC 选项，让返回的车辆信息按照距离这个中心位置从近到远的方式来排序，以方便选择最近的车辆；还可以使用 COUNT 选项，指定返回的车辆信息的数量。毕竟，5 公里范围内的车辆可能有很多，如果返回全部信息，会占用比较多的数据带宽，这个选项可以帮助控制返回的数据量，节省带宽。

可以看到，使用 GEO 数据类型可以非常轻松地操作经纬度这种信息。

虽然我们有了 5 种基本类型和 3 种扩展数据类型，但是有些场景下，我们对数据类型会有特殊需求，例如，我们需要一个数据类型既能像 Hash 那样支持快速的单键查询，又能像 Sorted Set 那样支持范围查询，此时，我们之前学习的这些数据类型就无法满足需求了。那么，接下来，我就再向你介绍下 Redis 扩展数据类型的终极版——自定义的数据类型。这样，你就可以定制符合自己需求的数据类型了，不管你的应用场景怎么变化，你都不用担心没有合适的数据类型。

## 如何自定义数据类型?

为了实现自定义数据类型，首先，我们需要了解 Redis 的基本对象结构 RedisObject，因为 Redis 键值对中的每一个值都是用 RedisObject 保存的。

我在第 11 讲中说过，RedisObject 包括元数据和指针。其中，元数据的一个功能就是用来区分不同的数据类型，指针用来指向具体的数据类型的值。所以，要想开发新数据类型，我们就先来了解下 RedisObject 的元数据和指针。

## Redis 的基本对象结构

RedisObject 的内部组成包括了 type,、encoding,、lru 和 refcount 4 个元数据，以及 1 个`*ptr`指针。

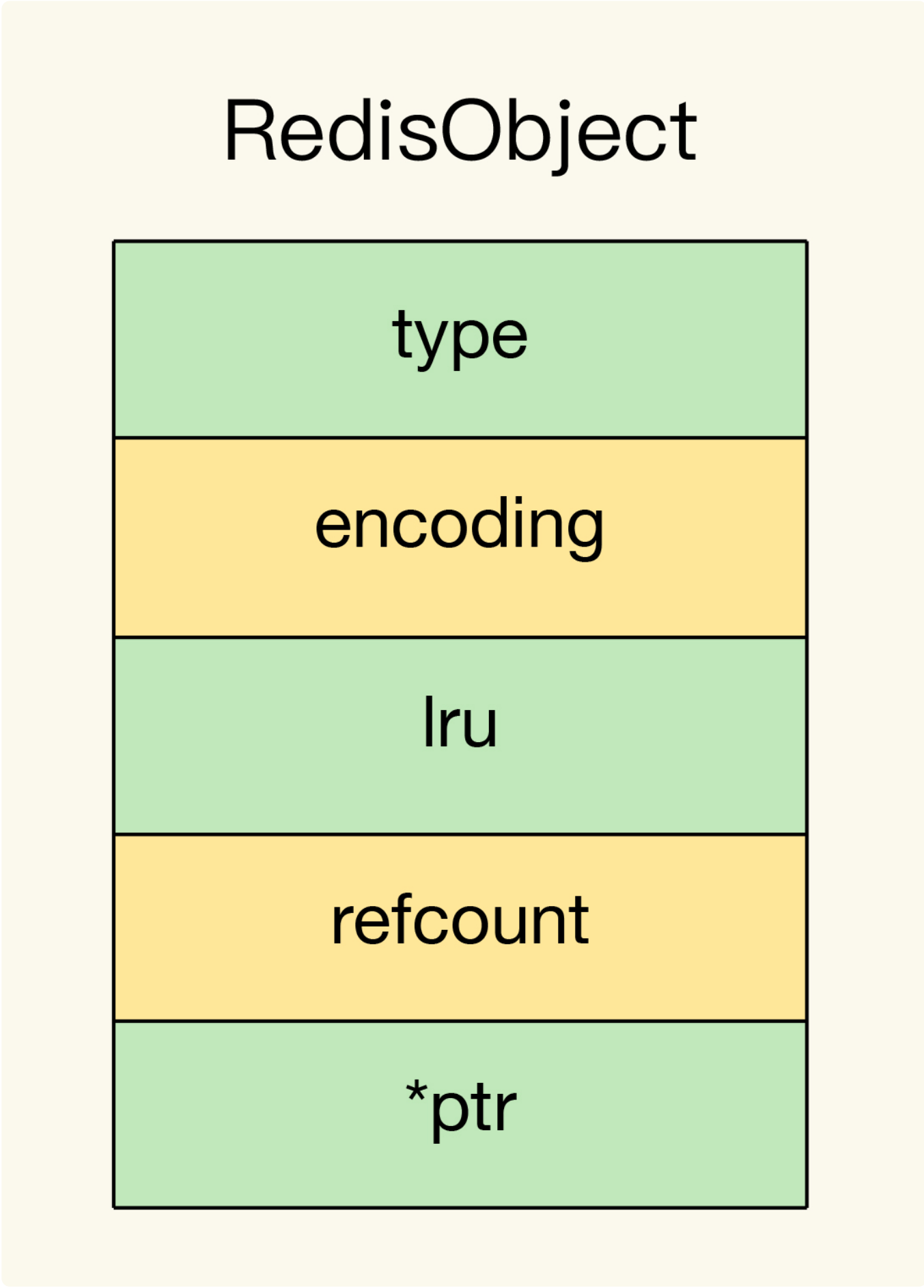
type：表示值的类型，涵盖了我们前面学习的五大基本类型；

encoding：是值的编码方式，用来表示 Redis 中实现各个基本类型的底层数据结构，例如 SDS、压缩列表、哈希表、跳表等；

lru：记录了这个对象最后一次被访问的时间，用于淘汰过期的键值对；

refcount：记录了对象的引用计数；

*ptr：是指向数据的指针。



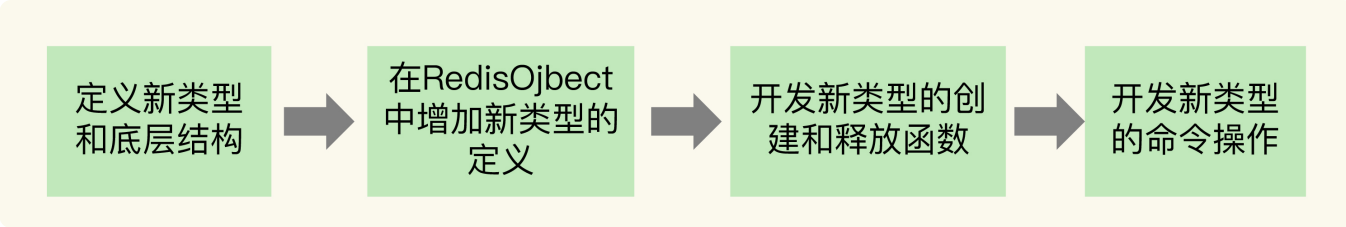

RedisObject 结构借助`*ptr`指针，就可以指向不同的数据类型，例如，`*ptr`指向一个 SDS 或一个跳表，就表示键值对中的值是 String 类型或 Sorted Set 类型。所以，我们在

定义了新的数据类型后，也只要在 RedisObject 中设置好新类型的 type 和 encoding，再用*ptr指向新类型的实现，就行了。

## 开发一个新的数据类型

了解了 RedisObject 结构后，定义一个新的数据类型也就不难了。首先，我们需要为新数据类型定义好它的底层结构、type 和 encoding 属性值，然后再实现新数据类型的创建、释放函数和基本命令。

接下来，我以开发一个名字叫作 NewTypeObject 的新数据类型为例，来解释下具体的 4 个操作步骤。




### 第一步：定义新数据类型的底层结构

我们用 newtype.h 文件来保存这个新类型的定义，具体定义的代码如下所示：

复制代码

```
struct NewTypeObject {
    struct NewTypeNode *head;
    size_t len;
}NewTypeObject;
```

其中，NewTypeNode 结构就是我们自定义的新类型的底层结构。我们为底层结构设计两个成员变量：一个是 Long 类型的 value 值，用来保存实际数据；一个是*next指针，指向下一个 NewTypeNode 结构。

复制代码

```
struct NewTypeNode {
    long value;
    struct NewTypeNode *next;
};
```

从代码中可以看到，NewTypeObject 类型的底层结构其实就是一个 Long 类型的单向链表。当然，你还可以根据自己的需求，把 NewTypeObject 的底层结构定义为其他类型。例如，如果我们想要 NewTypeObject 的查询效率比链表高，就可以把它的底层结构设计成一颗 B+ 树。

### 第二步：在 RedisObject 的 type 属性中，增加这个新类型的定义

这个定义是在 Redis 的 server.h 文件中。比如，我们增加一个叫作 OBJ_NEWTYPE 的宏定义，用来在代码中指代 NewTypeObject 这个新类型。

复制代码

```
#define OBJ_STRING 0    /* String object. */
#define OBJ_LIST 1      /* List object. */
#define OBJ_SET 2       /* Set object. */
#define OBJ_ZSET 3      /* Sorted set object. */
…
#define OBJ_NEWTYPE 7
```

### 第三步：开发新类型的创建和释放函数

Redis 把数据类型的创建和释放函数都定义在了 object.c 文件中。所以，我们可以在这个文件中增加 NewTypeObject 的创建函数 createNewTypeObject，如下所示：

复制代码

```
robj *createNewTypeObject(void){
    NewTypeObject *h = newtypeNew();
    robj *o = createObject(OBJ_NEWTYPE,h);
    return o;
}
```

createNewTypeObject 分别调用了 newtypeNew 和 createObject 两个函数，我分别来介绍下。

先说 newtypeNew 函数。它是用来为新数据类型初始化内存结构的。这个初始化过程主要是用 zmalloc 做底层结构分配空间，以便写入数据。

复制代码

```
NewTypeObject *newtypeNew(void){
    NewTypeObject *n = zmalloc(sizeof(*n));
    n->head = NULL;
    n->len = 0;
    return n;
}
```

newtypeNew 函数涉及到新数据类型的具体创建，而 Redis 默认会为每个数据类型定义一个单独文件，实现这个类型的创建和命令操作，例如，t_string.c 和 t_list.c 分别对应 String 和 List 类型。按照 Redis 的惯例，我们就把 newtypeNew 函数定义在名为 t_newtype.c 的文件中。

createObject 是 Redis 本身提供的 RedisObject 创建函数，它的参数是数据类型的 type 和指向数据类型实现的指针*ptr。

我们给 createObject 函数中传入了两个参数，分别是新类型的 type 值 OBJ_NEWTYPE，以及指向一个初始化过的 NewTypeObjec 的指针。这样一来，创建的 RedisObject 就能指向我们自定义的新数据类型了。

复制代码

```
robj *createObject(int type, void *ptr) {
    robj *o = zmalloc(sizeof(*o));
    o->type = type;
    o->ptr = ptr;
    ...
    return o;
}
```

对于释放函数来说，它是创建函数的反过程，是用 zfree 命令把新结构的内存空间释放掉。

**第四步：开发新类型的命令操作**

简单来说，增加相应的命令操作的过程可以分成三小步：

1. 在 t_newtype.c 文件中增加命令操作的实现。比如说，我们定义 ntinsertCommand 函数，由它实现对 NewTypeObject 单向链表的插入操作：

```
void ntinsertCommand(client *c){
  //基于客户端传递的参数，实现在NewTypeObject链表头插入元素
}
```

2. 在 server.h 文件中，声明我们已经实现的命令，以便在 server.c 文件引用这个命令，例如：

```
void ntinsertCommand(client *c)
```

3. 在 server.c 文件中的 redisCommandTable 里面，把新增命令和实现函数关联起来。例如，新增的 ntinsert 命令由 ntinsertCommand 函数实现，我们就可以用 ntinsert 命令给 NewTypeObject 数据类型插入元素了。

```
struct redisCommand redisCommandTable[] = {
...
{"ntinsert",ntinsertCommand,2,"m",...}
}
```

此时，我们就完成了一个自定义的 NewTypeObject 数据类型，可以实现基本的命令操作了。当然，如果你还希望新的数据类型能被持久化保存，我们还需要在 Redis 的 RDB 和 AOF 模块中增加对新数据类型进行持久化保存的代码，我会在后面的加餐中再和你分享。

## 小结

这节课，我们学习了 Redis 的扩展数据类型 GEO。GEO 可以记录经纬度形式的地理位置信息，被广泛地应用在 LBS 服务中。GEO 本身并没有设计新的底层数据结构，而是直接使用了 Sorted Set 集合类型。

GEO 类型使用 GeoHash 编码方法实现了经纬度到 Sorted Set 中元素权重分数的转换，这其中的两个关键机制就是对二维地图做区间划分，以及对区间进行编码。一组经纬度落在某个区间后，就用区间的编码值来表示，并把编码值作为 Sorted Set 元素的权重分数。

这样一来，我们就可以把经纬度保存到 Sorted Set 中，利用 Sorted Set 提供的“按权重进行有序范围查找”的特性，实现 LBS 服务中频繁使用的“搜索附近”的需求。

GEO 属于 Redis 提供的扩展数据类型。扩展数据类型有两种实现途径：一种是基于现有的数据类型，通过数据编码或是实现新的操作的方式，来实现扩展数据类型，例如基于 Sorted Set 和 GeoHash 编码实现 GEO，以及基于 String 和位操作实现 Bitmap；另一种就是开发自定义的数据类型，具体的操作是增加新数据类型的定义，实现创建和释放函数，实现新数据类型支持的命令操作，建议你尝试着把今天学到的内容灵活地应用到你的工作场景中。

## 每课一问

到今天为止，我们已经学习 Redis 的 5 大基本数据类型和 3 个扩展数据类型，我想请你来聊一聊，你在日常的实践过程中，还用过 Redis 的其他数据类型吗？

欢迎在留言区分享你使用过的其他数据类型，我们一起来交流学习。如果你身边还有想要自己开发 Redis 的新数据类型的朋友，也希望你帮我把今天的内容分享给他 / 她。我们下节课见。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 12 | 有一亿个keys要统计，应该用哪种集合？

下一篇 14 | 如何在Redis中保存时间序列数据？

## 精选留言 (13)

写留言

**Kaito**
2020-09-04

Redis也可以使用List数据类型当做队列使用，一个客户端使用rpush生产数据到Redis中，另一个客户端使用lpop取出数据进行消费，非常方便。但要注意的是，使用List当做队列，缺点是没有ack机制和不支持多个消费者。没有ack机制会导致从Redis中取出的数据后，如果客户端处理失败了，取出的这个数据相当于丢失了，无法重新消费。所以使用List用作队列适合于对于丢失数据不敏感的业务场景，但它的优点是，因为都是内存操作，...

展开

8 35

**Geek2014**
2020-09-10

想问一个扩展的问题，还请老师解答。

因为车辆是不断移动的，那如何维护车辆位置的GEOHASH呢。

展开

1

**晖**

2020-09-07

严格来说，根据Redis文档: "There are limits to the coordinates that can be indexed: areas very near to the poles are not indexable. Valid latitudes are from -85.05112878 to 85.05112878 degrees." 所以纬度的极值应该不能到正负90度。

展开

1

**小文同学**

2020-09-07

1、作者讲了什么?
GEO 的数据结构原理，及特点。还有编写 Redis 新数据结构和命令的步骤（此部分我没细看）。
2、作者是怎么把事情说明白的?
通过利用 LBS(Location-Based Service) 位置信息服务 来说引入 GEO 这个数据结构如...

展开

**东**

2020-09-07

"有的编码值虽然在大小上接近，但实际对应的方格却距离比较远。所以，为了避免查询不准确问题，我们可以同时查询给定经纬度所在的方格周围的 4 个或 8 个方格"。 可是这4个或8个里还是包含了这个有问题的方格，实践中如何剔除掉这个坏的方格呢?

展开

1

**王 慈**

2020-09-07

老师，这种通过更改源码实现自定义类型和功能的方式，比较好的处理redis版本更新的方法有什么呢？比如在v0.1稳定版上实现了功能，后续出现v0.2稳定版了是否可以升级呢?

**迷迷徒**

2020-09-07

geohash真是妙呀

展开

**d**

2020-09-06

老师，这块的代码改动能否给个fork url 看下?

展开

**MClink**

2020-09-05

老师后面会带我们去看源码吗?

展开

**一步**

2020-09-05

Redis 键值对中的每一个值都是用 RedisObject 保存

---

那么 redis 的键用什么保存的呢？ 也是 RedisObject 吗?

作者回复: 是的，Redis的键也是用RedisObject保存的。

1

**小袁**

2020-09-04

啥经纬度的位信息要交替存储，可以不交替存储么？原文只说了相邻编码可能在是相邻地区，但不相邻编码可以是相邻地区吧。既然这样，能否用算法直接算出相邻的八格的编码呢，然后直接根据权重查。

1

**马以**

2020-09-04

现在很多公司如果没有特殊场景，都是一个String走天下～

**那时刻**

2020-09-04

请问老师，redisobject里的 refcount：记录了对象的引用计数，这个对象引用计数在什么情况下发生呢？具体使用场景是什么？

展开